

SkipVis: A Skip Graph Visualizer

Yahya Hassanzadeh-Nazarabadi¹, Canberk Ekmekci¹, Esin Menceologlu², and Öznur Özkasap¹

¹Department of Computer Engineering, Koç University, Istanbul, Turkey

²Department of Computer Engineering, University of Minnesota, Minnesota, U.S.A

¹{yhassanzadeh13, cekmekci14, oozkasap}@ku.edu.tr

²mence007@umn.edu

Abstract—Skip Graph is a network overlay structure utilized in distributed data storage services as well as systems such as P2P cloud storage, search engines, Internet-of-Things, and mobile sensor networks. We propose *SkipVis*: a Skip Graph overlay visualizer that facilitates the flaw detection of Skip Graph-based algorithms in their design phase and also the implementation correctness verifiability. To the best of our knowledge, there exists no overlay visualizer for Skip Graph structure. Therefore, *SkipVis* is designed and developed as a scalable open source Skip Graph overlay visualizer that provides an interactive graphical user interface. In this paper, we describe *SkipVis* design principles, configuration, architecture and demonstration scenario.

Keywords—Distributed hash table, DHT, Skip Graph, P2P systems, visualization.

I. INTRODUCTION

Skip Graph [1] is a distributed hash table (DHT) data structure that keeps data objects in the form of (*key, value*) pairs where the *value* represents the data object and key corresponds to the hashed value of the data object. It is utilized as the communication structure for distributed data storage services [2], [3], due to its fast searching, load balancing, and search concurrency. Furthermore, Skip Graph is also employed as a distributed overlay in several applications including peer-to-peer (P2P) cloud storage [4], [5], [6], [7], [8], P2P search engines [9], [10], Internet-of-Things [11], and mobile sensor networks [12]. Likewise, Skip Graph can be used as a DHT alternative in various applications including DHT-based storage systems [13], DHT-based online social networks [14], and DHT-based search engines [15]. In such distributed overlays, a Skip Graph node commonly represents a connected device of the real world (e.g., laptop, cell phone, server, etc). Each Skip Graph node knows only a few other nodes of the system, and using this limited information, nodes can perform fully decentralized efficient searches for each other as well as each others' data objects.

The structure of a distributed Skip Graph overlay is directly affected by a wide variety of its operations including node arrivals (i.e., insertion) [1], node departures (i.e., deletion) [1], identifier assignment [16], and churn stabilization [17]. Such operations may distort the Skip Graph structurally and degrade its functionality. For example, a mistakenly inserted node by a faulty insertion protocol may mislead the search protocol of a Skip Graph to terminate with wrong result. Such an error can extend deeper and even make part of a Skip Graph unreachable from the rest. Design flaws can be detected and reported by implementing structural validation tests in a Skip Graph simulation environment (e.g., the Skip Graph simulator,

SkipSim [18]). The main disadvantage of this approach is that the resulted reports of failures are depicted based on the local view of the involved nodes. The local view of a Skip Graph node is its directly connected neighbors in the overlay network. Debugging a distributed Skip Graph protocol based on such discrete local views of a few involved nodes is timely inefficient and inaccurate. Likewise, some structural flaws are bi-product of others, and can not directly being detectable if their proper validation test is not implemented. For example, a wrong result of the search protocol that is detected by a search result validation test may be due to the existence of a loop in the Skip Graph overlay. Such a conclusion may not be directly extracted from the local views of the involved nodes on the search path. Also, in large scale overlays, for a considerable number of causes, including the cycle detection, executing a validation is not timely efficient.

In order to ease the detection of Skip Graph operational design flaws in the development phase and improve the implementation quality, we propose *SkipVis* [19], a Skip Graph overlay visualizer. *SkipVis* is a scalable open source Skip Graph overlay visualizer that provides an interactive graphical user interface (GUI). The *SkipVis*'s GUI enables the user to obtain a visual description of the input Skip Graph structure. Additionally, the user is able to dynamically interact with the depicted Skip Graph overlay, for example by inserting a new node. To the best of our knowledge, *SkipVis* is the only existing Skip Graph overlay visualizer. *SkipVis* can also be employed to serve the educational and instructional interests.

In the rest of this demo paper, we describe the structure of Skip Graph in Section II. We provide a sample demo scenario of *SkipVis* in Section III. The architecture of *SkipVis* is outlined in Section IV. We discuss the related works in Section V, followed by conclusion in Section VI.

II. PRELIMINARIES

Figure 1 illustrates a sample Skip Graph with 7 nodes, and 3 levels. In general, a Skip Graph with n nodes has $\lceil \log n \rceil$ levels that are numbered starting from 0. Each Skip Graph node is determined with two identifiers: a name ID and a numerical ID. Name IDs are binary strings of size $\lceil \log n \rceil$ bits and numerical IDs are non-negative integers. Each Skip Graph node has exactly one element at each level. In Figure 1, elements of the nodes are represented by squares with numerical IDs enclosed and name IDs located beneath each element. In level i of a Skip Graph there exists 2^i doubly linked-lists, and nodes that are located inside each list has at least i bits common prefix in their name IDs. Linked-lists

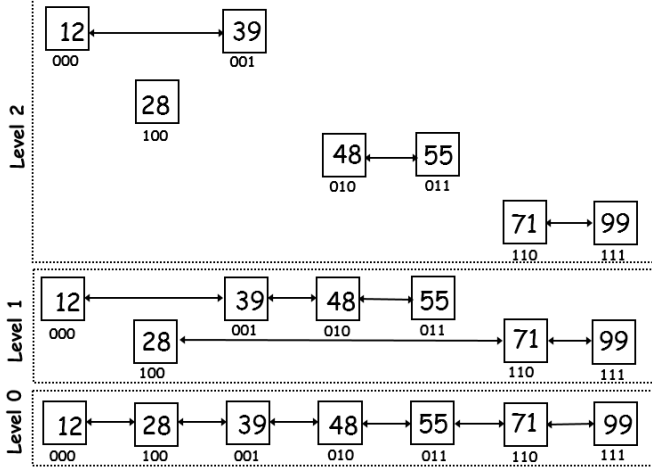


Fig. 1: A sample Skip Graph with 7 nodes and 3 levels. Numerical IDs of the nodes are enclosed in the squares with name IDs beneath.

are sorted based on the numerical IDs of their nodes in non-decreasing order.

As shown in Figure 1, the level 0 of a Skip Graph always constitutes of a single linked-list. In level 1 there exists two lists, one contains all the nodes with name ID prefix of 0, and the other one contains all the nodes with the name ID prefix of 1. Level 2 of the Skip Graph contains 4 doubly linked-lists corresponding to the 00, 01, 10, and 11 name ID prefixes. In the example of Figure 1 nodes with numerical IDs of 71 and 99 have the common prefix of 11 in their name IDs. Hence, they are located in the same lists up to and including level 2. Node 28 and 71, though, have the one-bit common prefix of 1 in their name IDs and located in the same lists at levels 0 and 1. Likewise, since there exists no node in the Skip Graph with name ID prefix of 10 except node 28, this node is located solely in a list at level 2. Nodes of a Skip Graph are able to exploit the overlay in order to perform searches for each others' numerical IDs [1] and name IDs [20] in fully decentralized manner. As the result of the search protocol, the address of the owner of the numerical ID or name ID of interest is returned. Having n nodes in a Skip Graph, a search for name ID or numerical ID is done by traversing $O(\log n)$ nodes in expectation with high probability.

III. SAMPLE DEMO SCENARIO

In *SkipVis*, a Skip Graph is defined by setting a lineup file, which is a text file containing the numerical ID, name ID, and neighbors of each node of the input Skip Graph. Each line of the lineup file represents one node of the input Skip Graph. The name ID and numerical ID of the node are determined by the first two elements of the corresponding line. The following pairs of elements in the line determine the numerical IDs of the left and right neighbors of the node at each level starting from level zero, respectively. In *SkipVis* we consider the numerical IDs as greater than zero integers. Hence, if the node does not have any left or right neighbor, corresponding elements for those are denoted by zero in the lineup file. More details and line-by-line description of the lineup file is given in *lineup.txt* of *SkipVis* distribution bundle [19].

Figure 2 shows the sample lineup file of the Skip Graph

```
000 12 0 28 0 39 0 39
100 28 12 39 0 71 0 93
001 39 28 48 12 48 12 0
010 48 39 55 39 55 0 55
011 55 48 71 48 0 48 0
110 71 55 93 28 93 0 99
101 93 71 99 71 99 28 0
111 99 93 0 93 0 71 0
```

Fig. 2: Sample lineup configuration file (i.e., *lineup.txt*) of the Skip Graph in Fig.1.



Fig. 3: *SkipVis* GUI after visualizing the input Skip Graph that is configured by Fig.2.

that is shown by Figure 1. The first line represents a node with name ID of 000 and numerical ID of 12. At level 0, the node has a right neighbor with numerical ID of 28. At level 1 and 2, it has a right neighbor with the numerical ID of 39. Since the numerical ID of the left neighbor at all the levels is denoted by zero, the node does not have a left neighbor. After lineup file is prepared, *SkipVis* can be executed to visualize the input Skip Graph. Figure 3 shows the resulted Skip Graph that *SkipVis* depicts based on the provided sample lineup file of Figure 2.

As shown in Figure 3, *SkipVis* discriminates the levels of Skip Graph by drawing all connector lines of each level with a distinct color. By the default settings, each Skip Graph node in *SkipVis* is represented by its numerical ID. The transition between the numerical IDs and name IDs is done by the *Switch All* button for all the nodes, or by clicking on each element of a node individually. Also, information about a node (e.g., name ID, numerical ID and its level) becomes available if the node is hovered over by mouse.

In addition to the visualization feature, *SkipVis* also provides dynamic node insertion operation for instructional purposes as well as convenient debugging at user interface level by using the *add* button. Figure 4 shows an example of dynamic insertion of a new node with the numerical ID of 123 and name ID of 101 to the Skip Graph of the Figure 3.

IV. ARCHITECTURE AND DEVELOPMENT

The core classes of *SkipVis* design are *SkipNode*, *SkipNodeDatabase*, and *Main*. Figure 5 represents the inter-class interactions. The interactions between classes are done by either parameter passing or hierarchical object declaration i.e., one

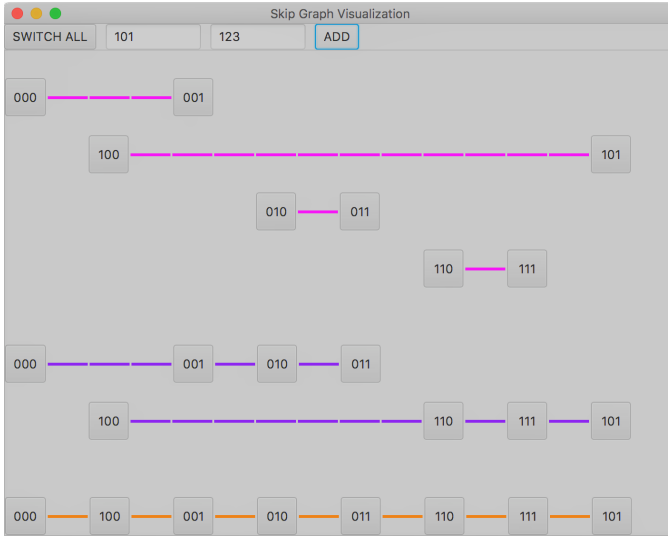


Fig. 4: The visualized Skip Graph of Figure 3 after dynamically inserting a new node with numerical ID of 123 and name ID of 101.

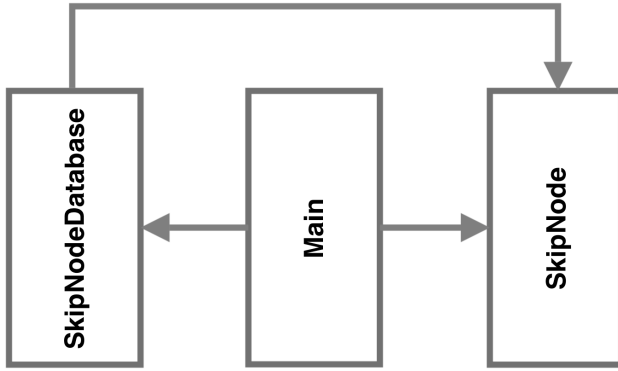


Fig. 5: Main classes of *SkipVis*

class object acts as the member variable of the other one. All interactions between classes are unidirectional. For example, in order to model a Skip Graph instance, *Main* class declares an *SkipNodeDatabase* object. *SkipNodeDatabase* class in turn implements functions like *insert* and *add* by declaring *SkipNode* objects. *add* method, as its name suggests, adds the *SkipNode* into *databaseArray.insert* method provides insertion of a new *SkipNode* object into *SkipNodeDatabase* by using *add* function. In addition, it also sets up all left and right neighbors of the *SkipNode* object. Also, *Main* class declares an *SkipNode* object to handle dynamic insertion event, which corresponds to the manual insertion of a node in the *SkipVis* by means of its GUI.

SkipNodeDatabase class represents a Skip Graph by hosting its set of nodes (i.e., *SkipNode* objects), number of Skip Graph nodes, and name ID length of nodes in the Skip Graph. Furthermore, *SkipNodeDatabase* class handles reading a Skip Graph description from the input lineup configuration file as well as dynamic insertion of nodes via GUI. *SkipNode* class represents a single Skip Graph node by its corresponding attributes e.g., name ID, numerical ID, neighboring information, etc. Additionally, *SkipNode* class contains validation tests

of the user's input data. For example, a negative numerical ID is invalidated at the constructor level by throwing an *InvalidIDException* exception.

Main class plays a vital role in *SkipVis*, since the main function is located in this class. *Main* class completely characterizes the GUI by having its containers (e.g. Scrollpane, VBox, HBox). There are four important method for visualization of the Skip Graph: *setScene*, *reset*, *nextLayer* and *create*. *setScene* method adjusts the scene by embedding VBox and HBox into the Scrollpane, and creates each layer by using *nextLayer* method. *reset* method erases the Skip Node structure in the Scrollpane, and replaces the Skip Graph with updated Skip Graph according to the *SkipNodeDatabase* object. *nextLayer* method creates the indicated layer of the Skip Graph according to the *SkipNodeDatabase*. Finally, *create* method takes *SkipNode* object and its level to visualize it as a Button object on *SkipVis*.

V. RELATED WORKS

To the best of our knowledge, there exists no overlay visualizer for Skip Graph structure. Although the Skip Graph simulator, *SkipSim* [18], visualizes geographical distribution of nodes in a Skip Graph-based P2P system, it does not provide any overlay visualization of the constructed Skip Graph. *Peersim* [21], which is a scalable P2P simulator, renders a geographical distribution overview of nodes with their neighboring relations. However, details like level, direction, and identifier, are missing in *Peersim*. *Oversim* [22], [23] is a scalable simulator framework for distributed overlays that also provides an overlay visualizer [24]. However, the main concentration of *Oversim* visualizer is on circular DHTs like Chord [13], [25], and there exists limited overlay visualization implementations of other types of DHTs e.g., GIA [26], EpiChord [27], and Pastry [28]. In contrast to *SkipVis* that aims to visualize the overlay's structure, *OverSim*'s visualizer module is topology oriented. In other words, while *SkipVis* illustrates how the overlay is affected by the nodes' identifiers, *OverSim* merely demonstrates the connections among the nodes with a lower focus on the overlay's structure. *PlanetSim* [29], [30] is another overlay simulator framework similar to the *Oversim* that provides implementations of Chord, Pastry, and Gnutella with an overlay visualizer available [31], [32]. However, in contrast to *SkipVis*, the *PlanetSim*'s visualizer module only represents the overall shape of the topology (e.g., circular, oval, star, etc) and does not present nodes' related information such as their identifier and neighboring relations.

A learning-oriented general-purpose framework of visualization is represented in [33], which helps to correlate the verbal and visual representation of the models. The framework aims at separating the application level from the visualization level to maximize the flexibility and minimize the overhead of visualization details. Additionally, to maximize the flexibility, the framework suggests discriminating the specification from the rendering as well employing a declarative representation. The specification of visualization could be coded, generated by user's mouse gestures or automatically specified by a program, which are aligned with the design aspects of *SkipVis*.

VI. CONCLUSION

We presented the configuration and architecture of our

proposed *SkipVis*, a Skip Graph [1] overlay visualizer. *SkipVis* [19] aims to facilitate the structural correctness verification and design flaws detection of the Skip Graph-based algorithms, and improve their implementation quality. *SkipVis* is also applicable as an instructional mean for Skip Graph. As future work, we plan to extend *SkipVis* by making it able to depict the Skip Graph search paths on arbitrary search queries that are given by the user. We also aim to enable *SkipVis* to inter-operate with the Skip Graph simulator, SkipSim [18], and monitor the structural effects of the events like churn on a Skip Graph overlay.

ACKNOWLEDGEMENT

The authors thank Türk Telekom and Koç University Summer Research Program (KUSRP) for their supports, and thank Deniz Yılmaz for her contributions to the *SkipVis* implementation.

REFERENCES

- [1] J. Aspnes and G. Shah, "Skip graphs," *ACM TALG*, 2007.
- [2] T. Shabeera, P. Chandran, and S. Kumar, "Authenticated and persistent skip graph: a data structure for cloud based data-centric applications," in *ACM CCS*, 2012.
- [3] S. T. Boshrooyeh and O. Ozkasap, "Guard: Secure routing in skip graph," in *IFIP Networking*. IEEE, 2017.
- [4] E. Udoh, *Cloud, Grid and High Performance Computing: Emerging Applications: Emerging Applications*. IGI Global, 2011.
- [5] S. Batra and A. Singh, "A short survey of advantages and applications of skip graphs," *IJSCE*, 2013.
- [6] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "Awake: decentralized and availability aware replication for p2p cloud storage," in *Smart Cloud*. IEEE, 2016.
- [7] Y. Hassanzadeh-Nazarabadi and Ö. Özkasap, "Elats: Energy and locality aware aggregation tree for skip graph," in *BlackSeaCom*. IEEE, 2017.
- [8] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "Decentralized and locality aware replication method for dht-based p2p storage systems," *Future Generation Computer Systems*, 2018.
- [9] A. Disterhöft, A. Funke, and K. Graffi, "Packetskip: Skip graph for multidimensional search in structured peer-to-peer systems," in *SASO*. IEEE, 2017.
- [10] P. Ganesan, B. Yang, and H. Garcia-Molina, "One torus to rule them all: multi-dimensional queries in p2p systems," in *SIGMOD/PODS*. ACM, 2004.
- [11] T. Toda, Y. Tanigawa, and H. Tode, "Autonomous and distributed construction of locality aware skip graph," in *CCNC*. IEEE, 2017.
- [12] G. J. Brault, C. J. Augeri, B. E. Mullins, R. O. Baldwin, and C. B. Mayer, "Enabling skip graphs to process k-dimensional range queries in a mobile sensor network," in *Network Computing and Applications*. IEEE, 2007.
- [13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM*, 2001.
- [14] S. Taheri-Boshrooyeh, A. Küpçü, and Ö. Özkasap, "Security and privacy of distributed online social networks," in *IEEE ICDCSW*, 2015.
- [15] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, 2003.
- [16] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "Locality aware skip graph," in *IEEE ICDCSW*, 2015.
- [17] R. Jacob, A. Richa, C. Scheideler, S. Schmid, and H. Täubig, "Skip+: A self-stabilizing skip graph," *Journal of the ACM*, 2014.
- [18] "Skipsim:<https://gitlab.com/yhassanzadeh13/skipsim-distribution-bundle>."
- [19] "Skipvis:<https://github.com/yhassanzadeh13/skipvis>."
- [20] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and O. Ozkasap, "Laras: Locality aware replication algorithm for the skip graph," in *IEEE/IFIP NOMS*, 2016.
- [21] A. Montresor and M. Jelasity, "Peersim: A scalable p2p simulator," in *Peer-to-Peer Computing*, 2009.
- [22] I. Baumgart, B. Heep, and S. Krause, "Oversim: A flexible overlay network simulation framework," in *IEEE Global Internet Symposium*, 2007.
- [23] —, "Oversim: A scalable and flexible overlay framework for simulation and real network applications," in *P2P*. IEEE, 2009.
- [24] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai, "A survey of peer-to-peer network simulators," in *The Seventh Annual Postgraduate Symposium*, vol. 2, 2006.
- [25] I. Baumgart, B. Heep, and S. Krause, "A p2psip demonstrator powered by oversim," in *P2P*. IEEE, 2007.
- [26] J. P. Munoz-Gea, J. Malgosa-Sanahuja, P. Manzaneres-Lopez, J. C. Sanchez-Aarnoutse, and A. M. Martinez-Rojo, "Simulation of a p2p application using oversim," in *Advances in Future Internet*. IEEE, 2009.
- [27] J. Furness, F. Chowdhury, and M. Kolberg, "An evaluation of epichord in oversim," in *NetCom2013*. Springer, 2014.
- [28] J. Furness, M. Kolberg, and M. Fayed, "An evaluation of chord and pastry models in oversim," in *Modelling Symposium*. IEEE, 2013.
- [29] P. García, C. Pairet, R. Mondéjar, J. Pujol, H. Tejedor, and R. Rallo, "Planetsim: A new overlay network simulation framework," in *International Workshop on Software Engineering and Middleware*. Springer, 2004.
- [30] J. Pujol-Ahullo and P. Garcia-Lopez, "Planetsim: An extensible simulation tool for peer-to-peer networks and services," in *P2P*. IEEE, 2009.
- [31] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers, "The state of peer-to-peer simulators and simulations," *ACM SIGCOMM Computer Communication Review*, 2007.
- [32] S. Naicken, A. Basu, B. Livingston, S. Rodhetbhai, and I. Wakeman, "Towards yet another peer-to-peer simulator," in *HET-NET*, 2006.
- [33] T. Piskuliyski and A. Kumar, "A general framework for overlay visualization," *Electronic Notes in Theoretical Computer Science*, 2007.